Homework 10 Selected Solutions
APPM/MATH 4650 Fall '20 Numerical Analysis

Due date: Saturday, November 21, before midnight, via Gradescope. **Instructor**: Prof. Becker
Theme: Multistep methods

solutions version 11/18/2020

Problem 1: In addition to the Adams-Bashforth, Adams-Moulton and Backward Differentiation formulas, there are other linear multistep methods, such as Milne's method (explicit), the midpoint method (explicit), and Simpson's method (implicit). We'll work with the ODE $y' = f(t, y)$.

a) Derive Simpson's *method* by applying Simpsons (quadrature) *rule* to the integral below (the integral coming from the FTC and the ODE)

$$y(t_{i+1}) - y(t_{i-1}) = \int_{t_{i-1}}^{t_{i+1}} f(t, y(t))dt.$$

Simpson's method is in our book Burden & Faires at the very end of chapter 5.6 (note: in some old editions of the text, there was a typo in the book here; it's fixed in the 9th and 10th editions).

Solution:

Approximate the integral on the right-hand-side by Simpson's rule which is (see chapter 4.3 in Burden and Faires)

$$\int_{t_0}^{t_2} f(t)\,dt = \frac{h}{3}\Big(f(t_0) + 4f(t_1) + f(t_2)\Big) - \frac{h^5}{90}f^{(4)}(\xi)$$

then we get the numerical scheme

$$\boxed{w_{i+1} = w_{i-1} + \frac{h}{3}\Big(f(t_{i-1}, w_{i-1}) + 4f(t_i, w_i) + f(t_{i+1}, w_{i+1})\Big)},$$

where $h = t_{i+1} - t_i$.

This looks a lot like an Adams-Moulton method, except that we have a w_{i-1} instead of a w_i on the right-hand-side.

b) What is the local truncation error $\tau(h)$? (The book gives the answer, but you should show how to get this; you *can* assume you know the error for Simpson's *rule*).

Solution:

The truncation error is obtained directly from the error in the Simpson's rule and the definition of truncation error.

$$\tau_{i+1}(h) = \frac{y_{i+1} - y_i}{h} - \frac{1}{3}\left[f(t_{i-1}, y_{i-1}) + 4f(t_i, y_i) + f(t_{i+1}, y_{i+1})\right] = -\frac{h^4}{90}f^{(4)}(\xi),\ \xi \in [t_{i-1}, t_{i+1}].$$

We didn't specify what to assume about $f^{(4)}$, but an answer saying $\boxed{\tau_{i+1}(h) = O(h^4)}$ was OK, as is the more specific $\boxed{-\frac{h^4}{90}f^{(4)}(\xi)}$ or $\boxed{-\frac{h^4}{90}y^{(5)}(\xi)}$ (since $y' = f$ so $y^{(5)} = f^{(4)}$).

Problem 2: a) Assume $y \in C^3[t_i, t_{i+2}]$ where $t_i \in \mathbb{R}$ is arbitrary and $t_{i+1} = t_i + h$ and $t_{i+2} = t_i + 2h$. Show that

$$y'(t_i) = \frac{-3y(t_i) + 4y(t_{i+1}) - y(t_{i+2})}{2h} + O(h^2) \tag{1}$$

and that

$$y'(t_{i+2}) = \frac{3y(t_{i+2}) - 4y(t_{i+1}) + y(t_i)}{2h} + O(h^2). \tag{2}$$

Bonus: show

$$y'(t_i) = \frac{-3y(t_i) + 4y(t_{i+1}) - y(t_{i+2})}{2h} + \frac{h^2}{3}y^{(3)}(\xi) \quad \text{for some } \xi \in [t_i, t_{i+2}]. \tag{3}$$

Solution:

Note that this is exactly the "three-point endpoint formula" given by Eq. (4.4) in Burden and Faires (chapter 4.1), so if students quoted that result directly, that's OK.

For the first equation, we'll show $-3y(t_i) + 4y(t_{i+1}) - y(t_{i+2}) = 2hy'(t_i) + O(h^3)$, which is equivalent (multiply the equation by $2h$ and re-arrange, noting that we generally take absolute values when using big-O notation, so the sign of $O(h^2)$ doesn't matter). Our key tool will be to Taylor expand $y(t_{i+1})$ and $y(t_{i+2})$ about $t = t_i$.

$$\begin{aligned}
-3y(t_i) + 4y(t_{i+1}) - y(t_{i+2}) &= -3y(t_i) + 4\left(y(t_i) + hy'(t_i) + \frac{h^2}{2}y''(t_i) + \frac{h^3}{6}y^{(3)}(\xi_1)\right) + \\
&\quad - \left(y(t_i) + 2hy'(t_i) + 4\frac{h^2}{2}y''(t_i) + \frac{4h^3}{3}y^{(3)}(\xi_2)\right) \\
&= (-3 + 4 - 1)y(t_i) + (4 - 2)hy'(t_i) + (4 - 4)\frac{h^2}{2}y''(t_i) \\
&\quad + 2\frac{h^3}{3}y^{(3)}(\xi_1) - \frac{4h^3}{3}y^{(3)}(\xi_2) \\
&= 2hy'(t_i) + O(h^3)
\end{aligned}$$

because both $y^{(3)}(\xi_1)$ and $y^{(3)}(\xi_2)$ are bounded by the extreme value theorem since $y^{(3)}$ is continuous.

To get the second formula, note that from the first formula we can write

$$y'(t_i) = \frac{-3y(t_i) + 4y(t_{i-1}) - y(t_{i-2})}{-2h} + O(h^2)$$

where we now go *backward* in time and so we just replaced $i+1$ with $i-1$ and change all h to $-h$. Now just adjust the index $i \leftarrow i + 2$ and multiply the numerator and denominator by -1 to get

$$y'(t_{i+2}) = \frac{3y(t_{i+2}) - 4y(t_{i+1}) + y(t_i)}{2h} + O(h^2)$$

Bonus: showing this is slightly subtle. We had an error term like $2\frac{h^3}{3}y^{(3)}(\xi_1) - \frac{4h^3}{3}y^{(3)}(\xi_2)$, but we *cannot* just declare that $\xi_1 = \xi_2$ and then combine these error terms to get $-\frac{h^2}{3}y^{(3)}(\xi)$. Instead, use interpolation, which is the approach used to derive Eq. (4.4) in Burden and Faires (chapter 4.1).

b) Consider the initial value problem (IVP)

$$y' = f(t, y) \quad \forall t \in [a, b], \qquad y(a) = \alpha.$$

Let n be an integer and $h = \frac{b-a}{n}$ and $w_0 = \alpha$. Eq. (1) suggests the difference method

$$w_{i+2} = 4w_{i+1} - 3w_i - 2hf(t_i, w_i) \qquad i = 0, 1, \ldots, n-2 \tag{4}$$

and Eq. (2) suggests the difference method

$$w_{i+2} = \frac{1}{3}\left(4w_{i+1} - w_i + 2hf(t_{i+2}, w_{i+2})\right) \qquad i = 0, 1, \ldots, n-2. \tag{5}$$

The method in Eq. (4) is an explicit method and doesn't have a name (that I know of). The method in Eq. (5) is the backward differentiation method BD2 (and is implicit).

For both methods, what is the local truncation error $\tau(h)$, and are these *consistent* methods?

Solution:

Both methods have local truncation error $\boxed{\tau(h) = O(h^2)}$ which follows from equations (2) and (1) and the definition of $\tau(h)$. Because $O(h^2) \to 0$ as $h \to 0$, this means both methods $\boxed{\text{are consistent}}$.

c) Analyze the *stability* and *convergence* for both methods. If the method is stable, specify if it is *strongly stable* or *weakly stable*.

Solution:

We check the root condition, which is motivated by the model equation $y' = \lambda y$. Then the characteristic polynomial (Eq. 5.58 in the book) to the explicit difference method Eq. (4) is

$$P(\lambda) = \lambda^2 - 4\lambda + 3, \text{ so } P(\lambda) = 0 \implies \lambda \in \{1, 3\} \implies \exists\, \lambda \text{ with } |\lambda| > 1.$$

(Recall that the $-2hf(t_i, w_i)$ term does not play a role here — one way to think about this is that we're concerned with the limit $h \to 0$ so we can ignore any term with an h). Therefore, the method is does not satisfy the root condition and hence is $\boxed{\text{unstable}}$. Hence by Theorem 5.24, since it is consistent and unstable, we conclude the method is $\boxed{\text{not convergent}}$. We'll see that our numerical results in the subsequent parts will confirm this. This is one reason that this method doesn't have a name — no one should be using it!

For the implicit BD2 method Eq. (5), the characteristic polynomial is

$$P(\lambda) = \lambda^2 - \frac{4}{3}\lambda + \frac{1}{3}, \text{ so } P(\lambda) = 0 \implies \lambda \in \left\{1, \frac{1}{3}\right\} \implies |\lambda| \le 1\, \forall \lambda$$

and $\lambda = 1$ is a simple root, so we satisfy the root condition, and are furthermore $\boxed{\text{strongly stable}}$ since there is no other root with $|\lambda| = 1$. Since we're consistent, by Theorem 5.24 the method is $\boxed{\text{convergent}}$. That's why this method has a name, because it's useful!

For the characteristic polynomial P, because we have a consistent scheme, we know $\lambda = 1$ is always a root, so if you don't find 1 as a root, then you've made a mistake.

d) Consider a specific IVP

$$y' = \cos(t) \quad \forall t \in [0, 0.1], \qquad y(0) = 0.$$

This is a very simple ODE and you should be able to work out the true solution by hand. Implement both the explicit method Eq. (4) and the implicit method Eq. (5) (BD2).

To check if your implementation is correct, when using $n = 10$ (i.e., $h = 10^{-2}$), the explicit method should give $w_n \approx 0.1097$ and the implicit method should give $w_n \approx 0.0998$. Report the values you get.

Details: As these are multi-step methods (and in particular, 2-step methods), we cannot use these methods to find w_1 since we'd need to have w_{-1}. So to find w_1, just evaluate the true solution at the relevant time (in a more realistic setting we would use something like RK4, but let's not involve RK4 for this problem).

For BD2, this is an implicit method, so we need to be able to solve a root-finding problem. However, because our ODE is so simple, this is not really a problem!

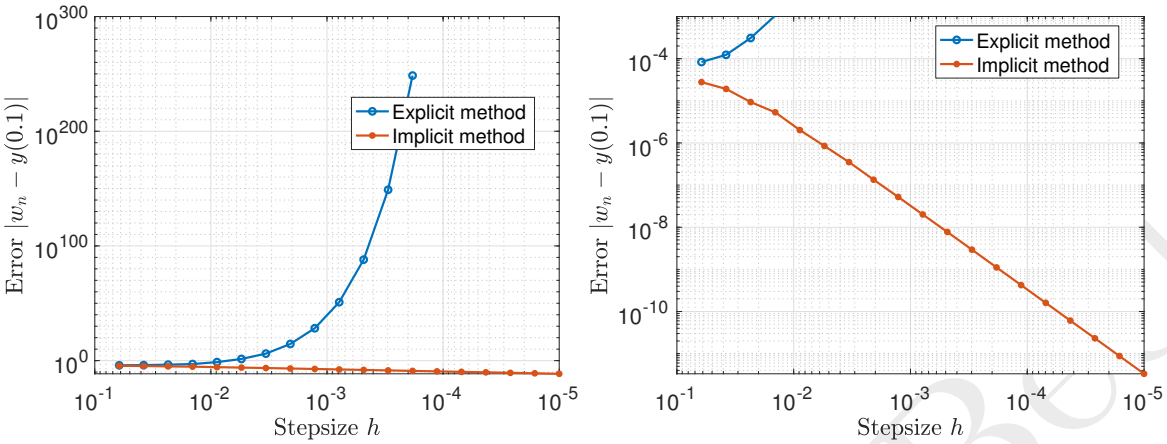

Figure 1: Plot for problem 2(e) (same plot, but different scalings on the y-axis)

Solution:

Some Matlab code:

```
tspan = [0,1e-1];
f     = @(t,y) cos(t);
y0    = 0;
trueSolution = @(t) sin(t);
h          = 1e-2;
[t,w] = HW10_reversedBD2( f, tspan, y0, h, trueSolution ); % Eq (4)
w(end) % 0.1097
[t,w] = HW10_BD2( f, tspan, y0, h, trueSolution ); % Eq (5)
w(end) % 0.0998
```

See the end of the PDF for the full code of `HW10_reversedBD2.m` and `HW10_BD2.m`.

e) For both methods, make a plot of the error $e = |w_n - y(0.1)|$ as a function of the stepsize h. For both methods, can you find a stepsize h that gives $e \leq 10^{-10}$? (If so, about what value is that h?)

Solution:

The plots are in Fig. 1. For the explicit method, we $\boxed{\text{cannot}}$ find a h to get $e \leq 10^{-10}$, and in fact we can't get any kind of error!! For the implicit method, we $\boxed{\text{can}}$ find such an h, e.g., about $\boxed{h \leq 5 \times 10^{-5}}$ is sufficient.

Code to make the plot:

```
hList = logspace(-1,-5,20);
errList_explicit = [];  errList_implicit = [];
for h = hList
    [t,w] = HW10_reversedBD2( f, tspan, y0, h, trueSolution );
    errList_explicit(end+1) = abs( w(end) - trueSolution(tspan(end)) );
    [t,w] = HW10_BD2( f, tspan, y0, h, trueSolution );
    errList_implicit(end+1) = abs( w(end) - trueSolution(tspan(end)) );
end
ind = ~isnan(errList_explicit);
figure(1); clf;
loglog( hList(ind), errList_explicit(ind),'o-', 'linewidth',2,'DisplayName',
    'Explicit method' )
hold all
```

```
loglog( hList, errList_implicit,'*-', 'linewidth',2 ,'DisplayName','Implicit
    method')
set(gca,'Fontsize',20); set(gca, 'XDir','reverse'); grid on
xlabel('Stepsize $h$','interpreter','latex');
ylabel('Error $|w_n - y(0.1)|$','interpreter','latex');
legend('location','best');
ylim([-Inf,1e-3]);
```

f) Comment on your findings

Solution:

There are a wide range of acceptable answers, but the elephant in the room is that the the explicit method is unstable and not convergent. No matter what kind of stepsize we try, we don't get good error.

The point of the problem is to give an explicit example that not all consistent methods are convergent. It's somewhat easy to derive a consistent method, but not all such methods are useful.

Problem 2 complete code

Here is the complete code. First, `HW10_reversedBD2.m`:

```
function [t,w] = HW10_reversedBD2( F, tspan, y0, h, trueSolution )
% Implements the reversed backward differentiation formula
% from HW 10. For simplicity, assumes a scalar not system of eq'n.
% This is an explicit method.
a = tspan(1); b = tspan(2);
n   = diff(tspan)/h;
t   = linspace(a,b,ceil(n)+1);
if length(t) < 2, error('choose smaller stepsize'); end
h   = t(2) - t(1);  % might not be exactly the same as original h

w   = zeros(1,length(t));
w(1)= y0;
f_i = F(t(1),y0);

% This is a multistep method, so we need to start it off
% We could do RK, but let's do something *simpler*
%   (requires cheating... we're assuming we know the true solution)
for i = 2:2
    w(i)  = trueSolution( t(i) );
    f_old   = f_i;
    f_i     = F( t(i), w(i) );
end

% Run the reversed BD2 code
for i = 3:length(t)

    w(i)  = 4*w(i-1) - 3*w(i-2) - 2*h*f_old;
    f_old   = f_i;
    f_i     = F( t(i), w(i) );

end
```

and `HW10_BD2.m`:

```
function [t,w] = HW10_BD2( F, tspan, y0, h, trueSolution )
% Implements the backward differentiation formula
% from HW 10. For simplicity, assumes a scalar not system of eq'n.
% This is implicit, but we're going to cheat, and use that we know
%   F is a function only of t, not of y
a = tspan(1); b = tspan(2);
n   = diff(tspan)/h;
t   = linspace(a,b,ceil(n)+1);
if length(t) < 2, error('choose smaller stepsize'); end
h   = t(2) - t(1);  % might not be exactly the same as original h

w   = zeros(1,length(t));
w(1)= y0;

% This is a multistep method, so we need to start it off
% We could do RK, but let's do something *simpler*
%   (requires cheating... we're assuming we know the true solution)
for i = 2:2
    w(i)  = trueSolution( t(i) );
end

% Run the reversed BD2 code
for i = 3:length(t)

    f_i     = F( t(i), [] ); % cheating
    w(i)  = ( 4*w(i-1) - w(i-2) + 2*h*f_i)/3;

end
```